Parallel Computing for Science & Engineering

3/21/2023

Instructors

Lars Koesterke, TACC

Victor Eijkhout, TACC

Quiz

Thursday, March 30

- During class time
- On Canvas
- Online/remote
- I will be on zoom to answer questions
- Designed for 45 minutes
- Quiz will stay open for 90 minutes

Outline (as of 3/21)

- This week OpenMP
- Next week Tuesday: OpenMP review
- Thursday: Quiz

- 1 lecture on OpenMP tasks
- 2 lectures on CUDA

- Homework will be posted tomorrow

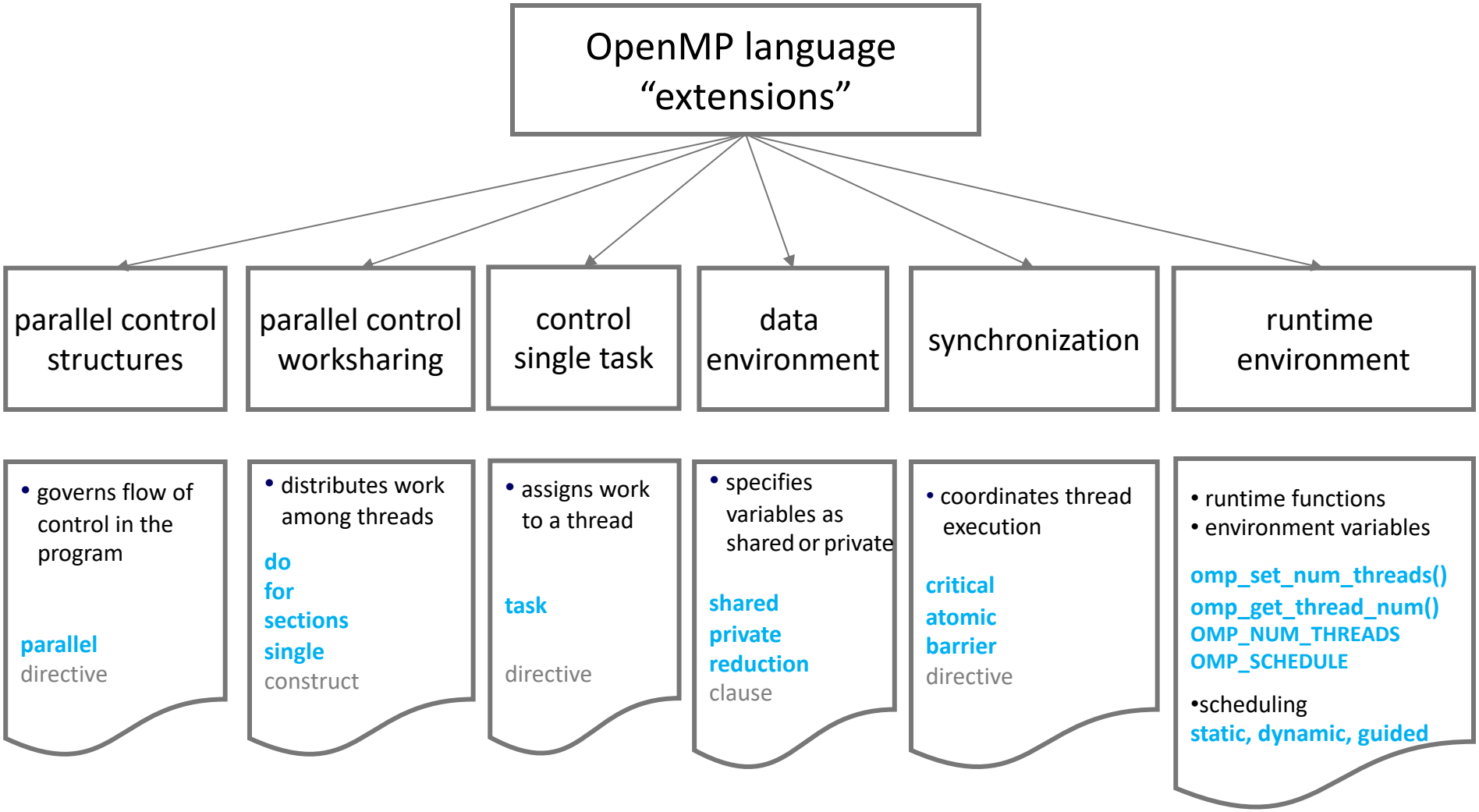
Use all resources that help

- Class material, books, etc.
- No talking to others!

Let me know if you cannot take the quiz on that day

- Slack, please

OpenMP Constructs



OpenMP Worksharing -- Single

- SINGLE (or MASTER)
  - Block of code is executed only once by a single thread (or the master thread)
  - Implied barrier (ONLY single)
- Only one thread is executing the block at all!

```
!$OMP single

  glob_count = glob_count + 1
  print *, glob_count

!$OMP end single
```

```
#pragma single
{
  glob_count++;
  printf("%d\n", glob_count);

}
```

Mutual exclusion – atomic and critical directives

When each thread must execute a section/statement of code serially (only one thread at a time can execute it) the region must be marked with CRITICAL/ATOMIC directives.

Use the ATOMIC directive if executing only one operation.

All threads are executing the block one after another

```
!$omp parallel shared(sum,x,y)
 ...
!$omp critical
   call update(x)
   call update(y)
   sum=sum+1
!$omp end critical
...
!$omp end parallel
```

```
 !$omp parallel
  ...
 !$omp atomic
       sum=sum+1
 ...
 !$omp end parallel
```

Syntax:

$pragma omp critical [*name*]

!$omp critical [*name*]

!$omp end critical [*name*]

TACC

Mutual exclusion – atomic and critical directives

When each thread must execute a section/statement of code serially (only one thread at a time can execute it) the region must be marked with CRITICAL/ATOMIC directives.

Use the ATOMIC directive if executing only one operation.

```
#pragma parallel shared(sum,x,y){
  ...
#pragma omp critical{
    update(x);
    update(y);
    sum=sum+1;
}
...
}
```

```
#pragma omp parallel{
  ...
#pragma omp atomic
        sum=sum+1;
...
}
```

Syntax:

$pragma omp critical [*name*]

!$omp critical [*name*]

!$omp end critical [*name*]

OpenMP Synchronization -- NOWAIT

Where a work-sharing region ends, a barrier is implied - all threads must reach the barrier before any can proceed. By using the NOWAIT clause at the end of each loop inside the parallel region, an unnecessary synchronization of threads can be avoided.

```
!$omp parallel
…

!$omp do
      do i=1,n
         call work(i)
      enddo
!$omp end do nowait

!$omp do schedule(dynamic,m)
      do i=1,n
         x(i)=y(i)+z(i)
      enddo

!$omp end parallel
```

```
#pragma omp parallel
{…

#pragma omp for nowait
      for(i=0; i<n; i++)
         work(i);



#pragma omp for schedule(dynamic,m)
    for(i=0; i<n; i++)
         x(i)=y(i)+z(i);


}
```


Dynamic scheduling is used to allow early threads to do more work in second loop."

OpenMP Worksharing Directives

- NOWAIT clause
  - Threads encounter a barrier synchronization at end of worksharing constructs.
  - Specifies that threads completing assigned work can proceed.

C/C++

Include as clause in C/C++.

```
#pragma omp for      nowait
#pragma omp single   nowait
#pragma omp sections nowait
```

F90

Fortran include on end statement

```
!$omp end do       nowait
!$omp end single   nowait
!$omp end sections nowait
```

Runtime Library API

API Environment Variables

OMP_NUM_THREADS	Set to No. of Threads
OMP_SCHEDULE	Schedule for the run-time clause

Example

```
export OMP_NUM_THREADS=4
export OMP_SCHEDULE='static,4'
./a.out
```

Runtime Library API

Functions	Operation
`omp_get_max_threads`	from OMP_NUM_THREADS or after `omp_set_num_threads()`
`omp_get_num_threads()`	Number of Threads in team,N.
`omp_get_thread_num()`	Thread ID. {0 -> N-1}
`omp_get_num_procs()`	Number of machine hardware threads
`omp_in_parallel()`	True if in parallel region & multiple thread executing
`omp_set_num_threads(#)`	Changes Number of Threads for parallel region.

API Example: Pseudo code

Assume: 'export OMP_NUM_THREADS=3'

```
print omp_get_num_procs()
print omp_get_max_threads()

omp parallel
  print omp_get_num_threads()
  print omp_get_thread_num()
omp end parallel

omp_set_num_threads(6)
print omp_get_num_procs()
print omp_get_max_threads()
Print omp_get_num_threads()


omp parallel num_threads(2)
  print omp_get_num_threads()
  print omp_get_thread_num()
omp end parallel
```

```
16
3


3 printed 3 times
0, 1, or 2 each printed once



?
?
?


?
?
```

TACC

API Example: Pseudo code

Assume: 'export OMP_NUM_THREADS=3'

```
print omp_get_num_procs()
print omp_get_max_threads()

omp parallel
  print omp_get_num_threads()
  print omp_get_thread_num()
omp end parallel

omp_set_num_threads(6)
print omp_get_num_procs()
print omp_get_max_threads()
Print omp_get_num_threads()


omp parallel num_threads(2)
  print omp_get_num_threads()
  print omp_get_thread_num()
omp end parallel
```

```
16
3


3 printed 3 times
0, 1, or 2 each printed once



16
6
1



2 printed twice
0, 1
```

API example

Or can use OPENMP pragma

```
#include <omp.h>

#pragma omp parallel private(i,n)
{
  i  = omp_get_thread_num( ) ;
  n = omp_get_num_threads( );

}
```

```
use omp_lib

!$omp parallel private(i,n)

      i = omp_get_thread_num( )
     n = omp_get_num_threads( )

!$omp end parallel
```

What happens if you don't compile with –qopenmp?

OpenMP Conditional Compilation

Fortran triggers: !$

```
i=0; n=1
!$omp parallel private(i,n)
!$     i = omp_get_thread_num( )
!$     n = omp_get_num_threads( )
       call sub(i,n)
!$omp end parallel
```

Or can use OPENMP pragma (Fortran: compile with -fpp)

```
#include <omp.h>
i=0; n=1;
#pragma omp parallel private(i,n)
{
  #ifdef  _OPENMP
      i = omp_get_thread_num( ) ;
     n = omp_get_num_threads( );
  #endif
      sub(i,n);
}
```

```
use omp_lib
i=0; n=1
!$omp parallel private(i,n)

 #ifdef  OPENMP
     i = omp_get_thread_num( )
    n = omp_get_num_threads( )
 #endif
    call sub(i,n)
!$omp end parallel
```

References

- Some material identical to:
  www.chpc.utah.edu/attachments/20110112.05/IntroOpenMP06.pdf
- This one is a real tutorial and even has test modules:
  http://www.citutor.org/login.php
- The sites
  www.llnl.gov/computing/tutorials/openMP/
  www.nersc.gov/assets/Uploads/XE62011OpenMP.pdf
  have good reference/tutorial pages for OpenMP.

Examples

```
export OMP_NUM_THREADS=2
./a.out

x and y : arrays with 2 elements
x(1) = 2.; x(2) = 4.
j = 0

!$omp parallel private(i)
!$omp do
do i=1, 2            ! Loop from 1 to 2
  j = j + 1
  y(i) = x(j)
enddo
!$omp end parallel
```

What is missing?

What values in y(1) and y(2)?

Examples

```
export OMP_NUM_THREADS=2
./a.out

x and y : arrays with 2 elements
x(1) = 2.; x(2) = 4.
j = 0

!$omp parallel private(i)
!$omp do
do i=1, 2              ! Loop from 1 to 2
!$omp critical
  j = j + 1
  y(i) = x(j)
!$omp end critical
enddo
!$omp end parallel
```

Examples

```
export OMP_NUM_THREADS=2
./a.out

x : array with 2 elements
x(1) = 2.; x(2) = 4.
sum = 0

!$omp parallel
!$omp do
do i=1 , 2
  sum = sum + x(i)
enddo
!$omp end parallel
```

What is missing?

Examples

```
export OMP_NUM_THREADS=2
./a.out

x : array with 2 elements
x(1) = 2.; x(2) = 4.
sum = 0

!$omp parallel reduction(+:sum)
!$omp do
do i=1 , 2
  sum = sum + x(i)
enddo
!$omp end parallel
```

Examples

```
export OMP_NUM_THREADS=4
./a.out

x : array with 8 elements
'all elements set to 1'
sum = 0

!$omp parallel reduction(+:sum)
!$omp do
do i=1 , 8
sum = sum + (x(i) * y(i) + fct(…))
enddo
!$omp end parallel
```

There can be more operations on the reduction line

Examples

```
export OMP_NUM_THREADS=4
./a.out

x(4,4) : array with 4x4 elements

!$omp parallel
!$omp do schedule(static,1) private(j)
do i=1, 4
  do j=1, i
    x(i,j) = x(i,j) + 1.
  enddo
enddo nowait

!$omp do schedule(static,1)
do i=1, 4
   …
enddo
```

What does the ‘nowait’ do?

Examples

```
export OMP_NUM_THREADS=4
./a.out

x(4,4) : array with 4x4 elements

!$omp parallel
!$omp do schedule(static,1) private(j)
do i=1, 4
  do j=1, i
    x(i,j) = x(i,j) + 1.
  enddo
enddo nowait

!$omp do schedule(static,1)
do i=1, 4
   …
enddo
```

What does the ‘nowait’ do?

But there is problem

Examples

```
export OMP_NUM_THREADS=4
./a.out

x(4,4) : array with 4x4 elements

!$omp parallel
!$omp do schedule(static,1) private(j)
do i=1, 4
  do j=1, i
    x(i,j) = x(i,j) + 1.
  enddo
enddo nowait

!$omp do schedule(dynamic,1)
do i=1, 4
   …
enddo
```

What does the ‘nowait’ do?

But there is problem

Examples
Collapse of loops

```
export OMP_NUM_THREADS=6
./a.out

x(4,4) : array with 4x4 elements

!$omp parallel
!$omp do schedule(static,1) private(j) collapse(2)
do i=1, 4
  do j=1, 10
    x(i,j) = x(i,j) + 1.
  enddo
enddo
```